CLAIMS

What is claimed is:

1. A computer-based method for compiling a source code file on a client computer, the source code file being stored on a remote server computer and being accessible via web protocols, the method comprising:

(a) accepting a manually specified compile command, the compile command including a set of parameters, the set of parameters including an identifier corresponding to the source code file;

(b) executing a compile procedure corresponding to the compile command, the compile procedure effecting conversion of the source code file into a file executable on the client computer,

wherein step (b) includes downloading the source code file from the remote server computer to the client computer using web protocols without executing a manually specified download command.

2. The computer-based method of claim 1 wherein the source code is "C" source code.

3. The computer-based method of claim 2 wherein the identifier corresponding to the source code file is a URL.

4. A computer-based method for executing an application on a client computer, the application functioning to process file data stored on a remote server computer, the file data stored on the remote server computer being accessible via web protocols, the method comprising:

(a) accepting a manually specified execute command, the execute command including a set of parameters, the set of parameters including an identifier corresponding to the file data;

(b) executing a procedure corresponding to the execute command, the procedure manipulating the file data on the client computer,

wherein step (b) includes downloading the file data from the remote server computer to the client computer using the web protocols without executing a manually specified download command.

5. The computer-based method of claim 4 wherein the identifier corresponding to the file data is a URL.

6. The computer-based method of claim 4 wherein the application is a compiler.

7. The computer-based method of claim 4 wherein the application is a word processor.

8. The computer-based method of claim 4 wherein the application is financial tracking software.

9. A computer system including a processor, memory associated with the processor, and a storage medium capable of storing a data file, the data file having a corresponding file identifier, the system comprising:

(a) an application software component comprised of instructions in the memory and executable by the processor, the application software component functioning to process the data file; and

(b) an I/O software component comprised of instructions in the memory and executable by the processor, the I/O software component functioning to accept the file identifier, to determine whether the file identifier is a URL and, if so, to retrieve the data file from a remote server using the file identifier and, if not, to retrieve the data file from the storage medium using the file identifier.

10. The computer system of claim 9 wherein source code corresponding to the I/O software component is included in a programming implementation I/O library stored on the storage medium.

11. The computer system of claim 9 wherein source code corresponding to the I/O software component is included in an Operating System I/O API stored on the storage medium.

12. The computer system of claim 11 wherein the Operating System is a Windows® operating system.

13. The computer system of claim 12 wherein the Operating System is a Windows® 2000 operating system.

14. The computer system of claim 13 wherein the storage medium is a hard disk drive.

15. The computer system of claim 14 wherein the application software component is a compiler component.

16. The computer system of claim 15 wherein the application software component is a word processing component.

17. The computer system of claim 16 wherein the application software component is a financial tracking component.

18. A computer-readable storage medium used in a computer system having a processor, memory associated with the processor and a storage device having a data

storage medium, the computer computer-readable storage medium having instructions capable of being executed by the processor for performing the following:

(a) accepting a file identifier corresponding to a data file; and

(b) determining whether the file identifier is a URL and, if so, retrieving the data file from a remote server using the file identifier and, if not, retrieving the data file from the data storage medium using the file identifier.

APPENDIX A

```
#ifndef IO_H
#define IO_H
#include <stdio.h>

/* $Id: io.h,v 1.1 2000/07/11 20:06:20Z drh Exp drh $ */

#define IO_T T
typedef struct T *T;

extern T IO_open(const char *file, const char *mode);
extern int IO_close(T stream);
extern int IO_flush(T stream);
extern int IO_getc(T stream);
extern int IO_putc(int c, T stream);
extern int IO_read(char *ptr, size_t size, size_t count, T stream);
extern int IO_write(char *ptr, size_t size, size_t count, T stream);

extern T IO_stdin;
extern T IO_stdout;
extern T IO_stderr;

#undef T
#endif


#define IO_T T

/* Standard file I/O */

struct file {
            struct T stream;
            FILE *fp;
};

static int fileclose(T stream) {
            FILE *fp = ((struct file *)stream)->fp;
            return fclose(fp);
}

static int fileflush(T stream) {
            FILE *fp = ((struct file *)stream)->fp;
            return fflush(fp);
}

static int fileread(char *ptr, size_t size, size_t count, T stream) {
            FILE *fp = ((struct file *)stream)->fp;
            return fread(ptr, size, count, fp);
}

static int filewrite(char *ptr, size_t size, size_t count, T stream)
{
```

```
                FILE *fp = ((struct file *)stream)->fp;
                return fwrite(ptr, size, count, fp);
}

static struct methods fileio = { fileclose, fileflush, fileread,
filewrite };

static T fileopen(const char *file, const char *mode) {
                FILE *fp = fopen(file, mode);
                if (fp) {
                                struct file *stream = malloc(sizeof *stream);
                                if (stream) {
                                                stream->stream.methods = &fileio;
                                                stream->fp = fp;
                                                return (T)stream;
                                }
                                fclose(fp);
                }
                return NULL;
}

static struct file
    stdinput  = { &fileio, stdin },
    stdoutput = { &fileio, stdout },
    stderror  = { &fileio, stdout };
T IO_stdin = (T)&stdinput, IO_stdout = (T)&stdoutput, IO_stderr =
(T)&stderror;
```

APPENDIX B

```
/* Net I/O */

#ifdef WIN32
#include <windows.h>
#include <wininet.h>

static HINTERNET hSession = NULL;

struct net {
                struct T stream;
                HINTERNET hFile;
                char buffer[128];
                char *bp, *limit;
};

static void netcleanup(void) {
                if (hSession)
                                InternetCloseHandle(hSession);
                hSession = NULL;
}

static int netclose(T stream) {
                HINTERNET hFile = ((struct net *)stream)->hFile;
                return InternetCloseHandle(hFile) == TRUE ? 0 : EOF;
```

```
}

static int netflush(T stream) {
        return EOF;
}

static int netread(char *ptr, size_t size, size_t count, T stream) {
        struct net *ns = (struct net *)stream;
        size_t n = count*size;
        if (ns->bp < ns->limit) {
                        for ( ; ns->bp < ns->limit && n >
0; n--)
                                *ptr++ = *ns->bp++;
                        return (count*size - n)/size;
        }
        if (InternetReadFile(ns->hFile, ptr, n, &count) == FALSE)
                count = 0;
        return count;
}

static int httpError(struct net *stream) {
        int count;
        char *bp = stream->bp = stream->limit = stream->buffer;
        if (!InternetReadFile(stream->hFile, stream->bp, sizeof
stream->buffer, &count))
                return 0;
        stream->limit = stream->buffer + count;
        for ( ; bp < stream->limit; bp++)
                if (*bp == '<' && (strncmp(bp, "<title>", 7)
== 0 || strncmp(bp, "<TITLE>", 7) == 0)) {
                        int code = 0;
                        for (bp += 7; bp < stream->limit
&& isspace(*bp); )
                                bp++;
                        while (bp < stream->limit &&
isdigit(*bp))
                                code = 10*code +
(*bp++ - '0');
                        if (code >= 401 && code <= 505)
                                return 1;
                        return 0;
                }
        return 0;
}

static T netopen(const char *file, const char *mode) {
        static struct methods netio = { netclose, netflush,
netread, nullwrite };
        HINTERNET hFile;
        if (hSession == NULL) {
                hSession = InternetOpen("",
INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
                if (hSession);
                        atexit(netcleanup);
```

```
                }
                if (strspn(mode, "RrbB") != strlen(mode))
                        return NULL;
                hFile = InternetOpenUrl(hSession, file, NULL, 0, 0, 0);
                if (hFile) {
                        struct net *stream = malloc(sizeof *stream);
                        if (stream) {
                                stream->stream.methods = &netio;
                                stream->hFile = hFile;
                                if (httpError(stream) == 0)
                                        return (T)stream;
                                IO_close((T)stream);
                                return NULL;
                        }
                        InternetCloseHandle(hFile);
                }
                return NULL;
        }
        #else

        static T netopen(const char *file, const char *mode) {
                return NULL;
        }
        #endif

        int IO_close(T stream) {
                int code;
                assert(stream);
                code = (*stream->methods->close)(stream);
                free(stream);
                return code;
        }

        int IO_flush(T stream) {
                assert(stream);
                return (*stream->methods->flush)(stream);
        }

        int IO_getc(T stream) {
                char c;
                assert(stream);
                if ((*stream->methods->read)(&c, 1, 1, stream) == 1)
                        return (unsigned)c;
                return EOF;
        }

        int IO_putc(int c, T stream) {
                char buf = c;
                assert(stream);
                if ((*stream->methods->write)(&buf, 1, 1, stream) == 1)
                        return c;
                return EOF;
        }
```

```
int IO_read(char *ptr, size_t size, size_t count, T stream) {
            assert(ptr);
            assert(stream);
            return (*stream->methods->read)(ptr, size, count,
stream);
}

int IO_write(char *ptr, size_t size, size_t count, T stream) {
            assert(ptr);
            assert(stream);
            return (*stream->methods->write)(ptr, size, count,
stream);
}


static int isUrl(const char *path) {
            return strstr(path, "://") != NULL;
}

T IO_open(const char *file, const char *mode) {
            const char *s;
            assert(mode);
            for (s = mode; *s; s++)
                        if (strchr("AaBbRrWw+", *s) == NULL)
                                    return NULL;
            if (file == NULL)
                        return nullopen(file, mode);
            else if (isUrl(file))
                        return netopen(file, mode);
            else
                        return fileopen(file, mode);
}
```